Semantic and multiple-view feature modelling: towards more meaningful product modelling

Willem F. Bronsvoort, Rafael Bidarra and Alex Noort

Computer Graphics and CAD/CAM Group
Faculty of Information Technology and Systems
Delft University of Technology
The Netherlands

W.F.Bronsvoort/R.Bidarra/A.Noort@its.tudelft.nl

Key words: Feature modelling, semantics, validity maintenance, multiple feature views

Abstract: Current feature modelling systems suffer from a number of shortcomings. One is that the meaning of features is often not adequately maintained during modelling, which implies that modelling is in essence only geometric modelling. Another is that it is not possible to support product models with multiple feature views for all product development phases.
This paper discusses solutions for these shortcomings. *Semantic feature modelling* is an approach to specify and maintain the meaning, or semantics, of each feature in a model. *Enhanced multiple-view feature modelling* is an approach to support more product development phases. In the views, feature models can be used that contain high-level information. These approaches result in more meaningful product modelling, both in the sense of the models that are created, and in the sense of how these are created.

1. INTRODUCTION

The most important way of product modelling is now feature modelling. Although the functionality of feature modelling systems has been considerably improved during the last decade, there are still shortcomings. This paper will discuss solutions for two of these shortcomings.

F. Kimura (ed.), *Geometric Modelling*

The first shortcoming is that the meaning of features is often not adequately maintained during modelling. In many current feature modelling systems, "features" only occur at the user interface level, whereas in the product model only the resulting geometry is stored. Such systems are in essence only geometric modelling systems. In other systems, see for example (Parametric 2000), information about features is stored in the product model, but it is not consistently checked that the meaning of all features is maintained during the whole modelling process. For example, a through hole can be turned into a blind hole by blocking one of the openings of the hole with a stiffener, without the system even notifying this change, see *Figure 1*. Although geometrically this is correct, it is incorrect in the sense that the meaning, or semantics, of the feature is changed from a through hole to a blind hole.

In the *semantic feature modelling* approach, the meaning, or *semantics*, of each feature in a model is adequately maintained (Bidarra and Bronsvoort 2000). In Section 2, it will be described how the semantics of all features is specified in their respective classes, using several types of constraints. In Section 3, it will be discussed how this semantics is maintained during all modelling operations.

Figure 1. Changing a through hole into a blind hole

The second shortcoming is that product models with multiple feature views for all product development phases are not yet possible. Current multiple-view feature modelling systems only support form feature views, which can, for example, be used for part detail design and part manufacturing planning. Views for conceptual design and assembly design are not yet supported.

In the *enhanced multiple-view feature modelling* approach presented here, views for all four product development phases mentioned above are possible. In the views, feature models can be used with high-level information. In Section 4, an overview will be given of current multiple-view form feature modelling systems. In Section 5, enhanced multiple-view feature modelling will be described, in particular the views that are possible and how these are related.

In Section 6, some conclusions and further developments will be discussed.

2. SPECIFICATION OF FEATURE CLASSES

Feature class specification involves specification of its shape, its positioning and orientation scheme, its validity conditions, and its interface, according to the general structure depicted in *Figure 2*. For all aspects, constraints are used. These *feature constraints* are members of the feature class, and are therefore instantiated automatically with each new feature instance.

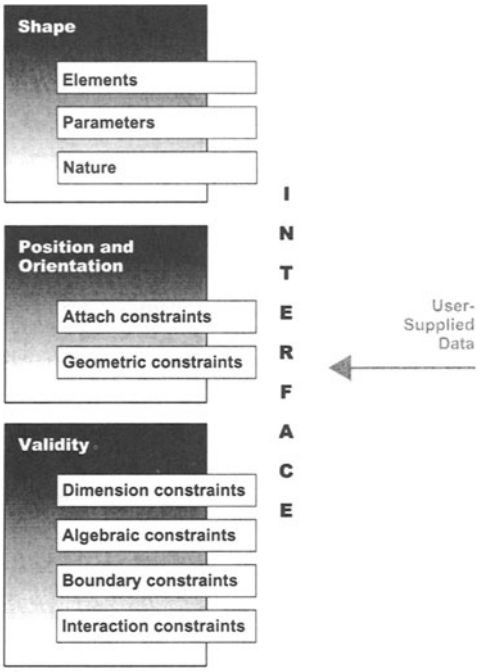

Figure 2. Feature class structure

The basis of a feature class is a parameterised shape. For a simple feature, this is a *basic shape*, e.g. a cylinder for a hole. A basic shape encapsulates a set of geometric constraints that relate its parameters to the corresponding shape faces. For a compound feature, the shape is a combination of several, possibly overlapping, basic shapes, e.g. two cylinders for a stepped hole. The faces of a feature shape are labelled with generic names, and these names can be used in all modelling operations. For example, a cylinder shape has a *top*, a *bottom* and a *side* face.

Associated to each feature is the notion of *nature*, indicating whether its shape represent material added to or removed from the model (respectively *additive* and *subtractive* natures).

The specification of validity conditions in a feature class can be classified into two categories: geometric and topologic.

One way of constraining the geometry of a feature, is by specifying the set of values allowed for a shape parameter. We use *dimension constraints* applied on shape parameters. For instance, the radius parameter of a through hole class could be limited to values between 1 and 10 *mm*. Another way of constraining the geometry of a feature is by means of explicit relations among its parameters. These relations can be simple equalities between two parameters (e.g. between width and length of a square passage feature) or, in general, algebraic expressions involving two or more parameters and constants. For this, we use *algebraic constraints*.

The set of shape faces of a feature provides full coverage of the feature boundary. However, for most features, not all faces are meant to effectively contribute to the boundary of the modelled product. Some faces, instead, have a closure role, delimiting the feature volume without contributing to the boundary. The specification of such properties is called *topologic validity* specification.

For this, we use two sorts of constraints: *boundary constraints* and *interaction constraints*. A boundary constraint states the extent to which a feature face should be on the model boundary. Boundary constraints are of two types: *onBoundary*, which means the shape face should be present on the model boundary, and *notOnBoundary*, which means the shape face should not be present on the model boundary. Furthermore, both types of boundary constraints are parameterised, stating whether the presence or absence on the model boundary is completely or only partly required. An example of this is a blind hole class for which the entrance face has a *notOnBoundary(completely)* constraint, the side face has an *onBoundary (partly)* constraint, and the bottom face has an *onBoundary(completely)* constraint.

Boundary constraints are unable to fully describe several other functional aspects that can be inherent to a feature class as well. These are better described in terms of the feature volume or feature boundary as a whole, instead of shape parameters or faces. An example of this is the requirement that every feature instance of some class should somehow contribute to the shape of the part model.

Such functional requirements can be violated by feature interactions caused during incremental editing of the model. *Feature interactions* are modifications of shape aspects of a feature that affect its functional meaning. An example of this is the *transmutation interaction* of the through hole into a blind hole in *Figure 1*. We propose the specification of *interaction constraints* in a feature class in order to indicate that a particular interaction type is not allowed for its instances (Bidarra *et al.* 1997).

Shape parameters and constraint variables of a feature may require *user-supplied data* to be provided at feature instantiation stage, as depicted in *Figure 2*. These parameters constitute the *feature class interface*. The specification of the feature class interface determines how feature instances will be presented to the user of the modelling system and, thus, how the user will be able to interact with them. Essential in the feature class interface is the positioning and orientation scheme, which is specified by means of attach and geometric constraints.

An *attach constraint* of a feature couples one of its faces to a user-supplied face of some feature already present in the model. For example, the top and bottom faces of a through hole may be used to attach it to, say, the top and bottom faces of a block, respectively.

Geometric constraints position and orient a feature relative to (faces of) other features in the model, by fixing its remaining degrees of freedom. For this, a geometric constraint relates one of the feature faces to a user-supplied feature face in the model, possibly with some additional numeric parameter(s). For instance, to position a through slot, a distanceFaceFace constraint might be used, which requires an external feature face and a distance value.

Some shape parameters may be implicitly determined from the feature attachments, e.g. the depth of a through hole. Other parameters may be explicitly specified via algebraic constraints. The remaining parameters need a user-supplied value at feature instantiation stage, and are therefore included in the feature class interface.

3. MODEL VALIDITY MAINTENANCE

Embedding validity criteria in each feature class, as described in the previous section, enhances the modelling process, as it guarantees that at the creation of a feature instance its semantics effectively matches the requirements of its class. In fact, one of the basic ideas of feature modelling is that functional information can be associated to shape information in a feature model. However, this association becomes useless when the shape imprint of a feature, once added to the model with a specific intent, is significantly modified later by a modelling operation. Stated differently, modifying the semantics of a feature should be disallowed if one wants to make feature modelling really more powerful than geometric modelling.

Feature model validity maintenance is the process of monitoring each modelling operation in order to ensure that all feature instances conform to the validity criteria specified for them. Maintaining feature model validity throughout the modelling process guarantees that all aspects of the design intent once captured in the model are permanently maintained.

Validity maintenance can be split into two types of tasks: (i) *validity checking*, performed at key stages of each modelling operation; and (ii) *validity recovery*, performed when a validity checking task detected a violation of some validity criterion.

Validity maintenance as described in this paper effectively raises the level of assistance provided by the feature modelling system. Together with the declarative feature class specification scheme presented in Section 2, it forms the core of the semantic feature modelling approach.

The approach has been fully implemented in the SPIFF system, a prototype multiple-view feature modelling system developed at Delft University of Technology (Bronsvoort *et al.* 1997).

The system maintains the *Feature Dependency Graph*, a high-level representation of the structure of the product. It contains all feature instances in the model, each of them with its own set of entities (shape, parameters and constraints), and all model constraint instances (i.e. constraints that are separately defined by the user, possibly between different features in the model, with the goal of further specifying design intent). The Feature Dependency Graph is a directed graph, defined by the set of all model entities (features and model constraints), and by the set of dependency relations among these entities. A feature f_1 is said to be dependent on a feature f_2 whenever f_1 is attached, positioned, or in some other way constrained relatively to f_2 (i.e. some feature constraint of f_1 has a reference to some entity of feature f_2).

The SPIFF system maintains also a geometric model of the product in the so-called *Cellular Model*, and takes care of updating it as required by each modelling operation (Bidarra *et al.* 1998). The Cellular Model is an evaluated representation of the feature model geometry, integrating the contributions from all features in the Feature Dependency Graph. The evaluated geometry of each feature, designated the feature's shape extent, accounts for the bounded region of space comprised by its volumetric shape. The Cellular Model represents a feature model as a connected set of volumetric quasi-disjoint cells, in such a way that each one lies either entirely inside a shape extent or entirely outside it. The cells in the Cellular Model represent the point sets of the shape extents of all features in the model. Each shape extent is, thus, represented in the Cellular Model by a connected subset of cells. The cellular decomposition is interaction-driven, i.e. for any two overlapping shape extents, some of their cells lie in both shape extents, whereas the remaining ones lie in either of them. In order to be able to search and analyse features, each cell has an attribute - called *owner list* - indicating which shape extents it belongs to, see *Figure 3*. Analogously, each cell face has also an owner list, indicating which shape faces it belongs to.

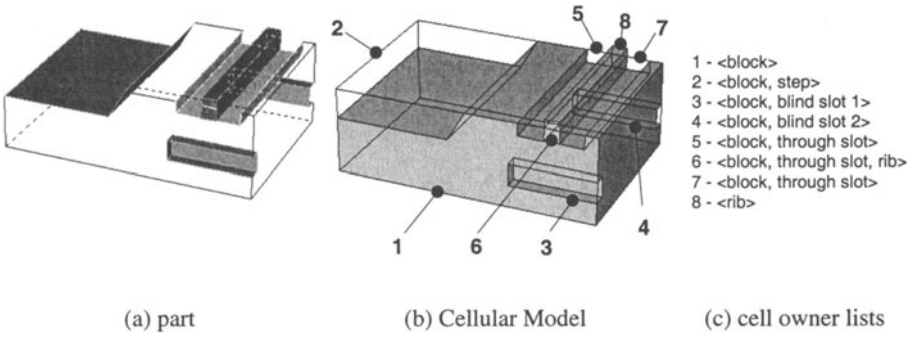

Figure 3. Cell owner lists in the Cellular Model

The basic idea of model validity maintenance is that a modelling operation, to be considered valid, should entirely preserve the design intent specified so far, with each feature as well as with all model constraints. In other words, after a valid modelling operation, the feature model conforms to all its constraints.

Modelling operations can be grouped into two major categories: feature operations and model constraint operations (or simply constraint operations). Feature operations include adding a new feature instance to the model, and editing or removing an existing feature in/from the model. Model constraint operations include adding, modifying and removing a model constraint.

The generic scheme of a modelling operation is presented in *Figure 4*, showing its main steps. Also shown in the diagram are the various points at which the operation can run invalid. When this is the case, the operation branches into the reaction loop instead of following the normal flow, and the model enters an invalid state. The main goal here is to enter the reaction loop, if required, with sufficient knowledge of the current status of the model, so that it can be appropriately handled, reported to the user and, ultimately, overcome. We now shortly describe the steps in the diagram:

1. **Dependency analysis** This step is only required for the removal of a feature from the model, which is not allowed if it has dependent entities (features or model constraints) in the Feature Dependency Graph.
2. **Interaction scope determination** The *feature interaction scope* (FIS) of a feature operation is the set of all feature instances in the model that may potentially be affected by the operation. It is used to avoid checking for feature interactions in vain later in the interaction detection (last step in *Figure 4*).
3. **Geometric and algebraic solving process** This step is required for all modelling operations, except feature removal. Its goal is to determine or update the dimensions, position and orientation of all features in the

model. The system deploys a dedicated constraint solver for each constraint type: a geometric constraint solver, based on extended 3D degrees of freedom analysis (Kramer 1992), and a SkyBlue algebraic constraint solver (Sanella 1992). At this stage, modelling operations are considered invalid if an *overconstrained* or *underconstrained situation* is detected.

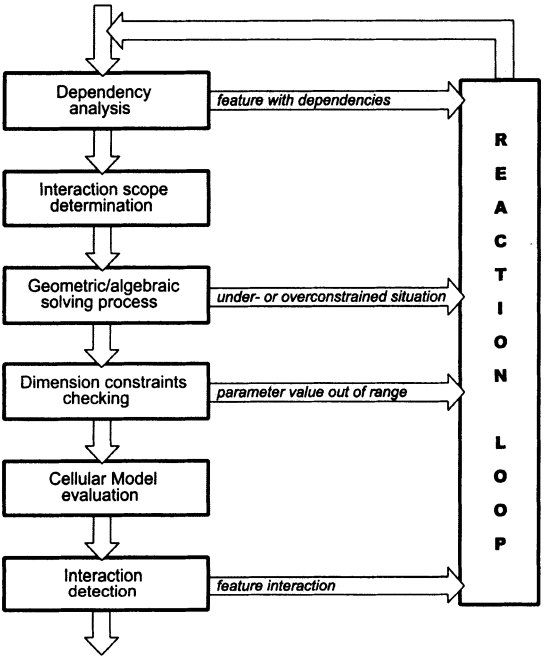

Figure 4. Generic scheme of a modelling operation

4. **Dimension constraints checking** When the solving process is successfully concluded, all feature shape dimensions have their values assigned, and checking of all dimension constraints takes place.
5. **Cellular Model re-evaluation** When this step is reached, each feature in the Feature Dependency Graph has all its parameters successfully updated. In particular, all feature shape extents have their dimensions, position and orientation fully determined. The Cellular Model can now be updated.
6. **Interaction detection** Once the Cellular Model has been updated, checking of topologic validity takes place. At this stage, a modelling operation is considered invalid if any boundary or interaction constraint is violated. See (Bidarra *et al.* 1997) for details on the interaction detection algorithms.

When a modelling operation is invalid, a valid model should be achieved again. This is straightforward if the modelling operation is cancelled: all that is needed is to backtrack to the valid model state just before executing it, by "reversing" the invalid operation.

However, to always have to recover from an invalid operation by undoing it is too rigid. It is often much more effective to constructively assist the user in overcoming the constraint violations, after an invalid modelling operation, in order to recover model validity again. In most cases, if the user receives appropriate feedback on the causes of an invalid situation, it is likely that other corrective actions might preferably be chosen. We call this process *validity recovery*, and it includes reporting to the user constraint violations, documenting their scope and causes, and, whenever possible, providing context-sensitive corrective hints.

To achieve this, a corrective mechanism was devised –the *reaction loop*, represented in *Figure 4*– which is activated whenever an operation turns out to be invalid. The user can then specify several modelling operations in a batch and execute them, in order to overcome the invalid model situation. Execution of these *reaction operations* follows the same scheme of *Figure 4*, which means that their outcome is checked for validity. The reaction loop is only exited when, as a result of the specified reactions, the model is valid again. At any stage when the model is invalid, the user may give up attempting to fix it and backtrack to the last valid stage.

The specification of reaction operations is supported by automatically generated hints, which document each constraint violation detected, and suggest solutions. These vary with the operation stage at which the reaction loop is entered, and with the type of constraint involved. Referring to the scheme of *Figure 4*, we have:

1. **Dependency analysis** At this point, the user is presented a list of all entities that depend on the feature *f* to be removed, in order to decide how to handle each of them. For example, the user might choose to remove with *f* some of its dependent entities, and to make others dependent on another feature.
2. **Geometric and algebraic solving process** For both over- and underconstrained situations, the reaction loop provides the user with a graphical notification of where the conflict was found, so that appropriate corrections can be made.
3. **Dimension constraints checking** The user is notified about the particular feature and parameter where the conflict was found, as well as about the admissible range for that parameter.
4. **Interaction detection** For each interaction detected, the user is graphically notified of its causes (mostly the features creating the interaction), and of its concrete effects (e.g. the feature face or parameter

affected). According to the particular interaction type, specific reaction choices may also be given (for example, after the modelling operation of *Figure 1*, the user is suggested to replace the through hole by a blind hole feature instance).

In all cases above, the scope of the reaction choices made available is restricted to those features and model constraints that are somehow involved in the invalid situation. This further assists the user in concentrating validity recovery efforts on effective and meaningful reactions.

4. MULTIPLE-VIEW FORM FEATURE MODELLING

Multiple-view form feature modelling is a product development approach that combines concurrent engineering and feature modelling. Concurrent engineering aims at designing better products in less time, by using Design for X (DFX—where X stands for any product life cycle phase) (Ulrich and Eppinger 2000) and by enabling simultaneous activities in several product development phases (Bullinger and Warschat 1995).

Multiple-view form feature modelling supports applications from various phases of product development, by providing interpretations of, or *views* on, the product model for each of these applications. Each view contains a form feature model specific for the application. Since the feature models of all views represent the same product, they have to be kept consistent. Quite a lot of research has been done on multiple-view form feature modelling during the last years. Some typical examples of this will be shortly discussed here.

De Martino *et al.* (1998) present a system architecture for form feature based modelling based on the integration of design by features and form feature conversion. The architecture allows creating and updating a design feature model, and deriving the context-oriented feature models for other applications from this design feature model. It uses one-way feature conversion: it only supports propagation of changes from the design feature model to the context-oriented feature models.

Hoffmann and Joan-Arinyo (2000) present an architecture for a product master model that connects a CAD system with applications from down-stream product development phases. The architecture allows the CAD system and the down-stream applications to deposit a representation of their internal model that is relevant for other applications in a central master model, and also allows these applications to associate information to elements of this central model. It uses what might be called partial multiple-way feature conversion: it supports propagation of all changes in the CAD model to the other applications, and propagation of minor changes from the applications back to the CAD system.

In the SPIFF modelling system (see Section 3), applications from several product development phases are integrated by providing views with a constraint-based form feature model for each of them (Dohmen *et al.* 1996), and combining these feature models into one product model. It uses multiple-way form feature conversion (de Kraker *et al.* 1997) to allow changes in the feature model of any view to be propagated to the feature models of the other views.

The approaches to multiple features views described above, which all use form features, share a number of shortcomings.

First, all approaches focus on the later product development phases, in which the geometry of the product has been fully specified. However, the early product development phases, such as conceptual design, are also very important, because the choices made in those phases have an enormous influence on the resulting product. In these phases, the geometry of the product is not yet completely known, so the use of form features is limited here, and more abstract features need to be introduced.

Second, all approaches deal with single parts. Real products, however, rarely consist of only a single part. Dealing with products that consist of multiple parts does not only involve dealing with the separate parts, but also with the relations between the parts. In particular, maintaining the validity of all parts and the relations between them can be very complex.

Third, all approaches discard the possibility that a feature model of a product for a view cannot be created, because the product does not satisfy all requirements from that view. In such situations, the model of the product should be adjusted in a way that the product satisfies the requirements for that view. To be able to adjust the model of the product automatically, the model should capture the intent of the designer in more detail than in current multiple-view feature modelling approaches.

5. ENHANCED MULTIPLE-VIEW FEATURE MODELLING

Instead of form feature models, enhanced feature models are used in the new multiple-view feature modelling approach discussed here, to support applications from more product development phases. Such a model is built from features, which are here defined as an aspect of the product that has some functionality. The features can be form features, but are in general features at a higher abstraction level. The prototype enhanced multiple-view feature modelling system is based on the SPIFF modelling system, and supports four product development phases; see *Figure 5*.

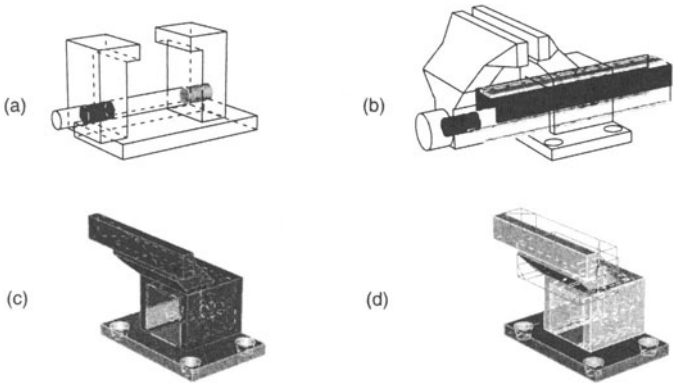

Figure 5. A conceptual design view (a), an assembly design view (b), a part detail design view (c), and a part manufacturing planning view (d), supported by the enhanced multiple-view feature modelling approach

The first phase, in which the product architecture is determined by specifying components and their interfaces, is conceptual design. Components are built from concepts, such as depressions and protrusions, and reference elements. Interfaces between components are specified by means of degrees of freedom between the components. The complete geometry of the components does not have to be specified in the conceptual design view. For example, for some concept only certain properties, such as its maximum volume, can be specified. An example of a conceptual design view for a bench vice, consisting of a base yaw, a moving yaw and a spindle, is given in *Figure 5*.a.

The second phase, in which the physical connections between the parts are determined, is assembly design. The connections are represented by connection features, such as dove-tail and pen-hole connection features; see also (van Holland and Bronsvoort 2000). An example of an assembly design view of the bench vice of *Figure 5.a*, with the form features for the connections between the components, is given in *Figure 5.b*.

The third phase, in which the details of the geometry of parts are determined, is part detail design. Detail design features are form features; examples are a through hole and a protrusion. An example of a part detail design view for the base yaw part of the bench vice of *Figure 5.a* is given in *Figure 5.c*.

The fourth phase, in which the way each part is to be manufactured is determined, is part manufacturing planning. Manufacturing planning features are again form features, such as slot and hole. An example of a manufacturing planning view for the base yaw part of the bench vice of *Figure 5.a* is given in *Figure 5.d*.

The new multiple-view feature modelling approach keeps the feature models of all views consistent, i.e. it ensures that all views represent the same product, based on the relations between these views.

Before discussing these relations in some detail, it should be noticed that they can be divided into a group that deals with the whole product, i.e. the conceptual design view and the assembly design view, and a group that deals with the individual parts, i.e. the part detail design views and the part manufacturing planning views. A part in one of the latter views corresponds to a single component in the first views; see *Figure 6*.

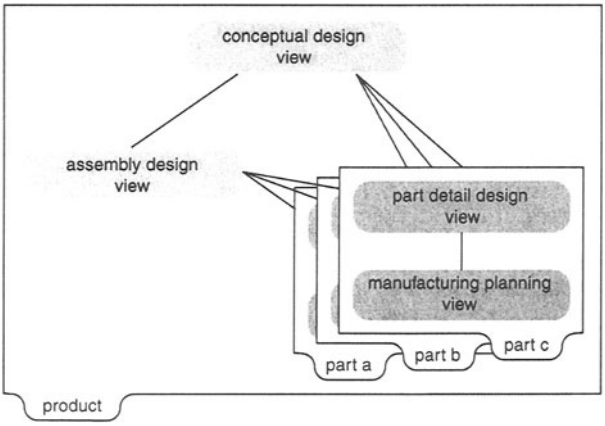

Figure 6. Relations between the supported views: the conceptual design view and the assembly design view deal with the whole product, whereas the part detail design views and the manufacturing planning views deal with the individual parts

The assembly design view allows the designer to refine the interfaces between the components in the conceptual design view. A connection feature needs to be created in the assembly design view for each interface in the conceptual design view, and linked to that interface. The interface and the connection feature should reduce the same freedom. In order to accommodate the connection feature, form features may be created on the components in the assembly design view. If the feature model of the assembly design view is changed, the feature models of the other views are updated in order to check whether their requirements are still satisfied.

The part detail design views allow the designer to refine the parts that are represented by the components in the conceptual design view, and which may have been refined in the assembly design view to accommodate connection features. Form features need to be created for each concept in the conceptual design view, and linked to that concept. The form features should satisfy the requirements specified for the concept, e.g. that the volume should be less than 80 cm^3. In addition, form features are automatically

created to represent the regions of a part that correspond to the form features of the connection features on the related component. If the feature model of a part detail design view is changed, the feature models of the other views are updated in order to check whether their requirements are still satisfied.

The manufacturing planning views allow the designer to analyse the parts for manufacturability and to create a manufacturing plan for them. The feature model in a manufacturing planning view is linked to the feature model in the corresponding part detail design view. Both views represent the same part, and should therefore have the same geometry. If the feature model of a manufacturing planning view is changed, the feature models of the other views are updated in order to check whether their requirements are still satisfied.

Feature conversion is used to keep the feature models of the views consistent, and thus to maintain the relations between the views. It involves linking features, mapping features and recognising features.

One of the ways in which incompletely specified geometry in a product model is supported, is the possibility to have variant and invariant parameters. Invariant parameters specify model characteristics that result from product requirements, variant parameters specify other model characteristics. Variant parameters can be changed without invalidating product requirements, invariant parameters cannot. Information on variant and invariant parameters can be used by the view conversion algorithm to automatically adjust the model, if no interpretation of the product model, in terms of features of that view, can be found such that all features are valid (Noort and Bronsvoort 1999). As described in Sections 2 and 3, feature validity is specified by constraints, and therefore, resolving a situation in which no interpretation of a product model is possible for some view, is essentially a constraint solving problem.

For the manufacturing planning view in *Figure 7.b*, no interpretation of the model represented by the part detail design view in *Figure 7.a* can be found, because two protrusions are too close together and a blind slot is too narrow for the resulting slots to be manufactured. In order to be able to build a consistent feature model for the manufacturing planning view with valid slot features, the model needs to be adjusted.

To be able to adjust the model in such a way that all specified product requirements remain satisfied, it first needs to be analysed. This analysis is based on the constraints of the product model, and uses techniques similar to techniques used to analyse under- and overconstrained models (Noort *et al.* 1998). This analysis takes into account that only constraints representing a variant parameter may be adjusted.

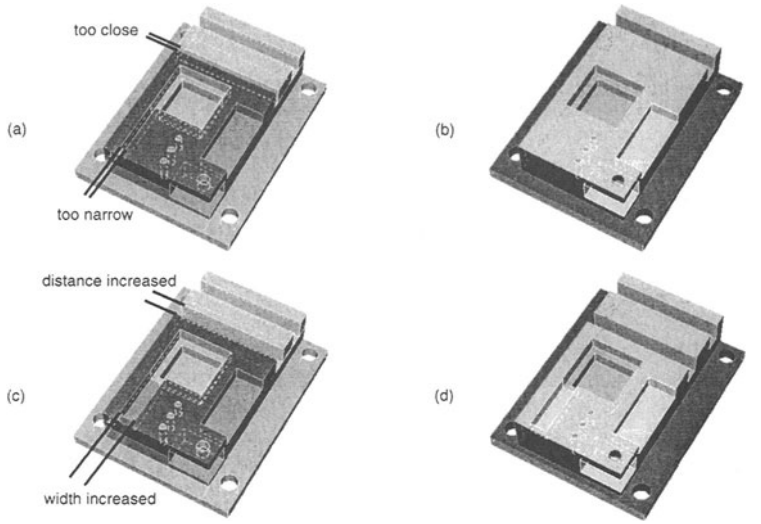

Figure 7. If no interpretation for some application can be found, a model is adjusted

After the analysis, the product model is adjusted, and additional features can be added to the model of the view. In the example, the distance between the two protrusions and the width of the blind slot are increased in the part detail design view (*Figure 7.c*), and as a result two valid slots can be created in the manufacturing planning view (*Figure 7.d*).

6. CONCLUSIONS

In the semantic feature modelling approach, the meaning of features has to be precisely defined in their respective classes, and this meaning is maintained during the whole modelling process.

Compared to more traditional geometric and feature modelling, the designer has less modelling freedom. On the other hand, it is guaranteed that only meaningful feature models are created. The designer is supported in this in a user-friendly way. This seems an important step forward to make feature modelling really more powerful than geometric modelling.

In the enhanced multiple-view feature modelling approach, views for a wider range of product development phases can be supported. It has been indicated how these views can be kept consistent.

A feature modelling system with the views described here can effectively support a major part of product development. Product information can be specified in meaningful ways, e.g. information on global properties of the product in the conceptual design view, and detailed information on relations between parts in the assembly design view.

In conclusion, the new feature modelling approaches described here result in more meaningful product modelling, both in the sense of the models that are created, and in the sense of how these are created.

REFERENCES

Bidarra, R. and Bronsvoort, W.F. (2000) Semantic feature modelling. *Computer-Aided Design* **32**(3): 201–225

Bidarra, R., Dohmen, M. and Bronsvoort, W.F. (1997) Automatic detection of interactions in feature models. In: *CD-ROM Proceedings of the 1997 ASME Design Engineering Technical Conferences*, 14–17 September, Sacramento, CA, USA, ASME, New York

Bidarra, R., de Kraker, K.J. and Bronsvoort, W.F. (1998) Representation and management of feature information in a cellular model. *Computer-Aided Design* **30**(4): 301–313

Bronsvoort, W.F., Bidarra, R., Dohmen, M., van Holland, W. and de Kraker, K.J. (1997) Multiple-view feature modelling and conversion. In: *Geometric Modelling: Theory and Practice - The State of the Art*, Strasser, W., Klein, R. and Rau, R. (eds), Springer, Berlin, pp 159–174

Bullinger, H.J. and Warschat, J. (1995) Concurrent simultaneous engineering systems. Springer-Verlag, London

De Martino, T., Falcidieno, B. and Hassinger, S. (1998) Design and engineering process integration through a multiple view intermediate modeller in a distributed object-oriented system environment. *Computer-Aided Design* **30**(6): 437–452

Dohmen, M., de Kraker, K.J. and Bronsvoort, W.F. (1996) Feature validation in a multiple-view modeling system. In: *CD-ROM Proceedings of the 1996 ASME Computers in Engineering Conference*, 19–22 August, Irvine, CA, USA, McCarthy, J.M. (ed), ASME, New York

Hoffmann, C.M. and Joan-Arinyo, R. (2000) Distributed maintenance of multiple product views. *Computer-Aided Design* **32**(7): 421–431

van Holland, W. and Bronsvoort, W.F. (2000) Assembly features in modeling and planning. *Robotics and Computer Integrated Manufacturing* **16**(4): 277–294

de Kraker, K.J., Dohmen, M. and Bronsvoort, W. F. (1997) Maintaining multiple views in feature modeling, In*: Solid Modeling '97, Fourth Symposium on Solid Modeling and Applications*, 14–16 May, Atlanta, GA, USA, Hoffmann, C.M. and Bronsvoort, W.F. (eds), ACM Press, New York, pp. 123–130

Kramer, G.A. (1992) Solving geometric constraint systems: a case study in kinematics. The MIT Press, Cambridge, MA

Noort, A. and Bronsvoort, W.F. (1999) Automatic model adjustment in form feature conversion. In: *CD-ROM Proceedings of the 1999 ASME Design Engineering Technical Conferences*, 12–16 September, Las Vegas, NV, USA, ASME, New York

Noort, A., Dohmen, M. and Bronsvoort, W.F. (1998), Solving over- and underconstrained geometric models. In: *Geometric Constraint Solving and Applications*, Brüderlin, B. and Roller, D., Springer-Verlag, Berlin, pp. 107–127

Parametric (2000) Pro/ENGINEER, *Version 2000i.* Parametric Technology Corporation, Waltham, MA

Sannella, M. (1992) The SkyBlue constraint solver. Technical Report 92-07-02, University of Washington, DC

Ulrich, K.T. and Eppinger, S.D. (2000) Product design and development. McGraw-Hill, New York